

Is “Sampling” better than “Evolution” for Search-based Software Engineering?

Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies, *Member, IEEE*

Abstract—Increasingly, SE researchers use search-based optimization techniques to solve SE problems with multiple conflicting objectives. These techniques often apply CPU-intensive evolutionary algorithms to explore generations of mutations to a population of candidate solutions. An alternative approach, proposed in this paper, is to start with a very large population and sample down to just the better solutions— but instead of evaluating all members of that population, just sample and evaluate pairs of distant examples. In studies with a dozen software engineering models, this sampling approach was found to be competitive with standard evolutionary algorithms (measured in terms of hypervolume and spread). Further, as software engineering models get more complex (e.g. heavily-constrained models of operating system kernels) this sampling approach performed as good or better than state-of-the-art evolutionary methods (and did so using up to 333 times fewer evaluations). That is, sampling algorithms is preferred to evolution algorithms for multi-objective optimization in software engineering domains where (a) evaluating candidates is very expensive and/or slow or (b) models are large and heavily-constrained.

Index Terms—Search-based SE, Sampling, evolutionary algorithms

1 INTRODUCTION

Software engineers often need to answer questions that explore trade-offs between competing goals. This is particularly true when stakeholders propose multiple goals or requirements and software developers need to find good choices that most reflect and balance rival objectives such as:

- 1) What are smallest set of test cases that cover all program branches?
- 2) What is the set of requirements that balances software development cost and customer satisfaction?
- 3) What sequence of refactoring steps take least effort while most decreasing the future maintenance costs of a system?

As modern software grows increasingly complex, it becomes difficult (or impossible) to manually find these good choices. Hence, in recent years, there has been an increasing interest in search-based software engineering, or SBSE (details see §2.1). SBSE often uses multi-objective evolutionary algorithms (MOEAs) [17], [33] that explore generations of mutations to a population of candidate solutions. Examples of this kind of analysis include:

- *Software product line optimization*: Sayyad et. al. [35] compared several MOEAs, including SPEA2, NSGA-II, IBEA, etc, and found that IBEA algorithm performed best in generating valid products from product line descriptions (for details see §4.2.3).
- *Project planning*: Ferrucci et al. [12] modified the crossover operator in the NSGA-II algorithm and found that their approach (called NSGA-II_p) was useful for planning how to make best use of project overtime.
- *Test suite minimization*: Wang et al. [44] showed that their “weighted-based” genetic algorithm significantly outperformed other methods using industrial case study for Cisco Systems.
- *Improving defect prediction*: Fu et al. [13] and Tantithamthavorn et al. [39] report that software quality predictors learned

from data miners can be improved if an evolutionary algorithm first adjusts the tuning parameters of the learner.

- *Software clone detectors*: Wang, Harman et al. [45] report that the arduous task of configuring complex analysis tools like software clone detectors can be automated via multi-objective evolutionary algorithms.

A drawback with standard MOEAs is that they can be very computational expensive (see §2.2). This can make them problematic to apply, especially for complex problems. For example, in the above list, the last two studies required 22 days and 15 years of CPU time respectively.

One way to address this CPU cost is to use cloud-based CPU farms. The advantage of this approach is that it is simple to implement (just buy the cloud-based CPU time). But cloud-based resources have some disadvantages: (a) cloud computing environments are extensively monetized so the total financial cost of tuning can be prohibitive; (b) that CPU time is wasted time if there is a faster and more effective way.

This paper asserts that there is indeed a faster and more effective way to solve multi-objective search-based SE problems. Specifically, we propose SWAY (short for the Sampling WAY). Unlike standard evolutionary programs, SWAY does not use mutation or perform any crossover between mutants. Rather it is a sampling technique that (a) inputs a large initial sample of candidate decisions then (b) outputs a small subset with best decisions. To understand the difference between SWAY and standard MOEAs:

- Standard MOEAs explore populations of, say, 100 candidates which they mutate and crossover for many generations.
- SWAY takes a population of 10,000 candidates then in a single generation, samples down to find the 100 best candidates.

If SWAY evaluated all 10,000 candidates, then it would be as slow as a standard MOEA. However, SWAY is a top-down clustering algorithm that (a) only evaluates $O(\log n)$ candidates (i.e. just the distant pairs of decisions at each level of recursion) then (b) recurses only on the data near the better candidates.

Hence, as shown in this paper, SWAY can terminate very quickly, even for large models.

This paper compares SWAY to standard MOEA algorithms in order to answer the following research questions.

RQ1: Can SWAY find optimizations as good as other MOEAs? Prior research by others has shown that MOEAs, such as IBEA, NSGA-II or SPEA2, etc. are useful for multi-goal optimization of SE models. Can our ultra-rapid alternative find competitive optimizations? We will use two widely-adopted metrics in SBSE research community—hypervolume and spread (see §4.3) to answer this question. Our results show that SWAY can find optimizations competitive with other MOEAs for the following models:

- XOMO [25], [27], [28], an SE model where the optimization task is to reduce the defects, risk, development months, and total number of staff associated with a software project.
- POM3 [3], [5], [32], an SE model of agile development teams negotiating what tasks to do next. The optimization task here is to deliver functionality of most value, at least cost.
- Models of software product lines [33], [35] from which the optimization task is to extract (a) valid products that use (b) the most familiar features that (c) cost the least to implement and which (d) has fewest known bugs.

Using these models, we show that:

Answer 1

SWAY can find the optimizations as good as other MOEAs.

RQ2: To what extent is SWAY faster than the MOEAs? The following case studies report the ratio

$$R = \frac{\#evals(other)}{\#evals(Sway)}$$

i.e. the number of evaluations required by other algorithms vs those required by SWAY. For this study, we use standard MOEAs that are commonly utilized within the SBSE community (including two that are arguably state of the art within their problem area: IBEASEED [33] and SATIBEA [18]). In the experiments, R values of $14 \leq R \leq 333$, with a median value of $R = 36$. That is:

Answer 2

SWAY requires one to two orders of magnitude less evaluations than existing methods.

RQ3: Is SWAY only applicable to small-size problems? SWAY just samples the space of solutions. Does this mean that this algorithm is only applicable to problems with a small, every simple, space of decisions? To test this, we applied SWAY to some very complex models—specifically, the software product line described above. Some of these product lines models are very complex: the biggest one studied here has nearly 7000 choices that are linked together by over 300,000 constraints. For these models, we found that SWAY is competitive with standard MOEAs. Further, as models grew more complex (e.g. the larger product line models) SWAY was seen to perform better than state-of-the-art evolutionary methods. That is:

Answer 3

SWAY can apply to large-size problems.

Based on the above, we conclude that sampling algorithms like SWAY are preferred to evolution algorithms for multi-objective optimization in software engineering domains where (a) evaluating

candidates is very expensive and/or slow or (b) models are large and heavily-constrained.

1.1 Relation to Prior Work

This paper significantly extends prior work of the authors. In 2014, Krall & Menzies proposed GALE [21], [22], [24], an evolutionary multi-objective optimizer that applied a novel mutation/crossover to recursively bi-cluster populations along the principal component of the decision space. GALE has two major drawbacks, which are addressed by this current paper. Firstly, while SWAY can generate useful optimizations for heavily-constrained large models, GALE could never succeed on those kinds of models. Secondly, this current paper shows that half of GALE was unnecessary. We know this since, when porting GALE from Python to Java, we accidentally disabled evolution. To our surprise, that “broken” version of GALE worked as well, or better, than the original GALE. This is an interesting result since GALE has been compared against dozens of models in a recent TSE article [21] and dozens more in Krall’s Ph.D. thesis [23]. In those studies, it was discovered that GALE was competitive against widely-used evolutionary algorithms— which lead to the conjecture that

The success of EAs is due less to evolution than to sampling many options.

SWAY was developed to test this conjecture. In 2016, a preliminary report on SWAY was presented at SSBSE’16 [40]— but that report just included results from the XOMO and POM3 models¹. While an intriguing early result, it turns out that methods that work for simple models like XOMO and POM3 do not work for more complex models of software product lines. In order to create an effective sampling algorithm for large & heavily constrained software product lines, we had to invent the radial pruning methods (as integrating SWAY into a SAT solver to generated the initial population from which it extracts its solutions). With the above changes, we can now offer a new high water mark in the processing complex SBSE models.

In summary, the unique contributions of this paper are:

- 1) A radical new approach to multi-objective optimization; i.e. use (a) sampling instead of (b) evolving, mutating, and crossing-over candidates.
- 2) A novel multi-objective optimization algorithm that implements that new approach.
- 3) A significant simplification and clarification of prior results concerning the GALE algorithm;
- 4) Double the number of case studies than explored in the prior SWAY paper [40];
- 5) These case studies show that combining (a) sampling; (b) radial pruning; and (c) a slightly smarter pre-processor allows for the very rapid processing of complex and large heavily constrained models.

In terms of textual material, the following sections of this paper have not appeared before: §3.2, the third research questions in §4.1 and §4.2.3.

1.2 Structure of this Paper

The rest of this paper is structured as follows. Section 2 introduces backgrounds for SWAY framework as well as its related domain materials. Section 3 details our SWAY framework. The research

1. Available on-line at <https://goo.gl/jn3p0Y/>

questions, benchmarks and experiment configurations are set up in section 4. Section 5 are the results for the case studies. More discussions and conclusions are in section 6 and 7.

2 BACKGROUND

2.1 Search Based Software Engineering (SBSE)

Throughout the software engineering life cycle, from requirement engineering, project planning to software testing, maintenance and re-engineering, software engineers need to find a balance between different goals such as:

- At what time should what features be released first to satisfy customer needs while, at the same time, best support future releases?
- How to staff a software project, so that the development team can reduce the cost estimation and shorten the releasing gap between versions?
- The four optimization tasks listed in the *Introduction*.

All of these problems can be viewed as *optimization problems*; i.e. tune the configuration parameters of a model such that, when that model runs, it generates “good” output (i.e. output demonstrably better than other possible output). However, given the complexities of software engineering, SE models are often too complicated to prove that an output is optimal. For such models, the best we can do is run multiple optimizers and report the best output seen across all those optimizers.

In the past, due to the simplicity of software structure, developers/experts could make a decision based on their empirical knowledge. For such models of such simple knowledge, it may have been possible to demand that those outputs are “optimal”; i.e. there exist no other configuration such that a better output can be generated. However, modern software is becoming increasingly complex. Finding the optimal solution to such kind of problems may be difficult/impossible. For example, in a project staffing problem, if there are 10 experts available and 10 activities to be accomplished, the total number of available combinations is 10 billion (10^{10}). For such large search spaces, exhaustively enumerating and assessing all possibilities is clearly impractical.

When brute force methods fail, it is possible to employ heuristics to explore complex models. The *Search Based Software Engineering (SBSE)*’s favorite heuristic is *metaheuristic search* algorithms such as genetic algorithms [20]. Such metaheuristics are “a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity” [2]. As seen in Figure 1, this approach has become increasing popular.

One advantage of these meta-heuristics is that they can simultaneously explore multiple goals at the same time. Next section introduces multi-objective evolutionary optimization algorithms, which are widely used in SBSE.

2.2 Multi-Objective Evolutionary Algorithm (MOEA)

In SBSE, the software engineering problem is treated as a mathematical model: given the numeric (or boolean) configurations/decisions variables, the model should return one or more objectives. In a nutshell, the model can convert *decisions* “*d*” into *objective* scores “*o*”, i.e.

$$o = model(d)$$

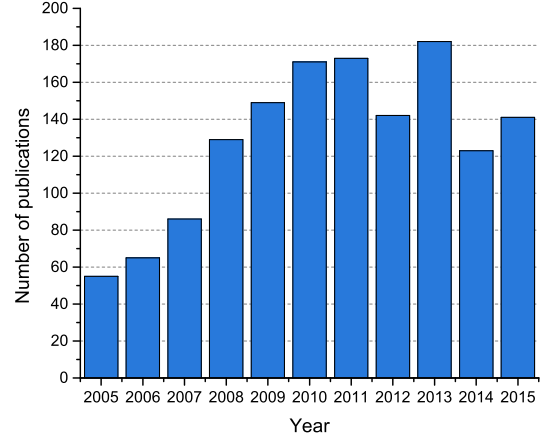


Fig. 1. The trend of publications on SBSE [48]

The direction of optimization for the objectives can be either to maximize or minimize their values. For example, in software engineering, we might want to maximize the delivered functionality while also minimizing the cost to make that delivery. If model delivers just one objective, then we call this a *single-objective optimization problem*. On the other hand, when there are many objectives we call that a *multi-objective optimization problem*.

For the multi-objective optimization problem, often there is no “*d*” which can minimize (or maximize) all objectives. Rather, the “best” *d* offers a good trade-off between competing objectives. In such a space of competing goals, we cannot be optimal on all objectives, simultaneously. Rather, we must seek a *Pareto frontier* or solution of multiple solutions where no other solutions in the frontier “dominates” any other [51].

There are two types of dominance— *binary dominance* and *continuous dominance*. Binary dominance is defined as follows: solution *x* is said to binary dominate the solution *y* if and only if the objectives in *x* is partially less (larger when the corresponding objective is to maximize) than the objectives in *y*, that is,

$$\forall o \in obj \ o_x \geq o_y \text{ and } \exists o \in obj \ o_x > o_y$$

where *obj* are the objectives and ($\geq, >$) tests if an objective score in one individual is (no worse, better) than the other.

Continuous dominance, as defined by [49], favors *y* over *x* if *x* “losses” least:

$$\begin{aligned} x > y &= loss(y, x) > loss(x, y) \\ loss(x, y) &= \sum_j^n -e^{\Delta(j, x, y, n)} / n \\ \Delta(j, x, y, n) &= w_j(o_{j, x} - o_{j, y}) / n \end{aligned} \quad (1)$$

MOEAs create the initial population first, and then execute the crossover and mutation repeatedly until “tired or happy”; i.e. until we have run out of CPU time limitation or until we have reached solutions that suffice for the purposes at hand. The basic framework for MOEAs is as follows:

- 1) Generate population *i* = 0 using some *initialization policy*.
- 2) Evaluate all individuals in population 0.
- 3) Repeat until tired or happy
 - a) *Cross-over*: combine elite items to make population *i* + 1;
 - b) *Mutation*: make small changes within population *i*;
 - c) *Evaluate*: individuals in population *i*;

d) *Selection*: choose some elite subset of population i .

One simple way to understand MOEAs is to compare them with Darwin’s theory of evolution. To find good scores for the objectives, start from a group of individuals. As time goes by, the individuals inside the group crossover. The offspring which have better fitness scores tend to survive (in the selection step). During the evolution, the mutation operation can increase the diversity of the group and avoid the evolution from getting trapped in the local optimal.

Two important details within MOEAs are the *initialization policy* in step (1) and the *evaluation policy* in step (3c). The standard initiation policy is to build members of the population by selecting, at random, across the range of all known decisions. For heavily-constrained models, this standard policy can be naive. For example, when we generated 10,000 random decisions using this standard initialization policy, the for the larger software product lines, less than 0.03% of the randomly generated solutions satisfied the domain constraints. Later in this paper, we discuss another policy; i.e. use a SAT solver to build an initial population.

As to the *evaluation policy*, the standard policy is, for each decision, run the underlying model to generate objective scores for those decisions. For some models, such an evaluation policy is confusing, prohibitively expensive, or both:

- Verrappa and Letier warn that “...for industrial problems, these algorithms generate (many) solutions, which makes the tasks of understanding them and selecting one among them difficult and time consuming” [43].
- Zuluaga et al. comment on the cost of evaluating all decisions for their models of software/hardware co-design: “synthesis of only one design can take hours or even days.” [52].
- Harman comments on the problems of evolving a test suite for software: if every candidate solution requires a time-consuming execution of the entire system: such test suite generation can take weeks of execution time [15].
- Krall & Menzies explored the optimization of complex NASA models of air traffic control. After discussing the simulation needs of NASA’s research scientists, they concluded that those models would take three months to execute, even utilizing NASA’s supercomputers [22].

Hence, later in this paper, we explore other *evaluation policies* that only evaluate a small percent of the population.

In practice, there are many MOEAs. They differ in the implement of selection, mutation, or crossover operations. Some widespread used MOEAs are as follows:

- *GAs = genetic algorithms*: GA models decisions as string of numbers (or binary symbols). To mate (crossover) two strings, just simply switch part of the strings inside two candidates [1], [20].
- *IBEA = the indicator-based evolutionary algorithm*: IBEA is a GA that uses the continuous domination function to prune away worst candidates [49];
- *NSGA-II = nondominated sorting genetic algorithm*: NSGA-II is a GA that uses a non-dominating sorting procedure to divide the solutions into *bands* where $band_i$ dominates all of the solutions in $band_{j>i}$. NSGA-II’s elite sampling favors the least-crowded solutions in the better band [9].
- *SPEA2 = Strength Pareto Genetic Algorithm, version 2*: SPEA2 is a GA that favors individuals that dominate the most number of other solutions that are not nearby (and to break ties, it favors items in low density regions) [50].

Algorithm 1: SWAY

Input : items – The candidates
Output : pruned results
Parameter : enough – The minimum cluster size
Require Func : SPLIT, see §3.1, §3.2
 BETTER, see §3.3

```

1 if numberOf (items) < enough then
2   return items
3 else
4    $\Delta_1, \Delta_2 \leftarrow \emptyset, \emptyset$ 
5   [west, east], [westItems, eastItems]  $\leftarrow$  SPLIT(items)
6   if  $\neg$ BETTER(west, east) then  $\Delta_1 \leftarrow$  SWAY(eastItems) if  $\neg$ BETTER(east,
   west) then  $\Delta_2 \leftarrow$  SWAY(westItems) return  $\Delta_1 + \Delta_2$ 
7
```

- And others such as MOEA/D [47], differential evolution [38], particle swarm optimization [30], and many more besides.

All the above algorithms typically evaluate thousands to millions of individuals as part of their execution. The intention of SWAY is to reduce that running time without sacrificing the quality of results.

3 SWAY, THE SAMPLING WAY

SWAY, short for the Sampling WAY, is a multi-objective optimizer. Unlike the MOEAs described above, SWAY does not execute over generations of mutated examples. Since evaluating all decisions might be time-consuming, SWAY recursively SPLITs the candidates into parts, then only evaluates and compares selected representatives of each parts (pruning away the candidates nearer to the worst representatives).

Algorithm 1 shows the general framework of SWAY:

- If the population size is smaller that some threshold, then we just return all candidates (line 1). Otherwise, SWAY splits the candidates into two parts, “west side” and the “east side”
- After that, lines 6 and 7 compares representatives of the sides. SWAY uses different methods to find those representatives—see §3.1 and 3.2.
- Prune the candidates based on a comparison of the representatives. If neither representative is better, then we SWAY on each part.

SWAY is a divide-and-conquer process. Let the number of candidates be n , the number of function calls to SPLIT be S_p . Then we have,

$$S_p(n) = 1 + kS_p\left(\frac{n}{2}\right) \quad (2)$$

where $k = 1$ (if only one of line 6 or 7 condition statement returns true) and $k = 2$ otherwise. According to the Master Theorem [7], $S_p = S_w = O(n)$ if k always is 2, and $S_p = O(\log n)$ if k always is 1. In our experience, $k = 2$ is very rare so SPLIT’s running time is $O(S_p) \approx O(\log n)$.

3.1 SPLIT for continuous decision spaces

SPLIT clusters the candidate into parts then picks up representatives for each part. For models with continuous decisions, we use the Fastmap heuristic [11], [31] to quickly split the candidates. Platt [31] shows that FastMap is a Nyström algorithm that finds approximations to eigenvectors.

Algorithm 2 lists the details of SPLIT function. To split the candidates into two parts according to the FastMap heuristic, first pick any random candidate (line 1) and then find the two extreme candidates based on the distances (line 2-3). The DISTANCE used in our case studies is the *Euclidean distance*. All other

Algorithm 2: SPLIT for continuous space (uses FASTMAP)

```

Input      : items – The candidates to split
Output     : [west, east] – representatives;
               [westItems, eastItems] – two parts
Require Func : DISTANCE

1 rand ← randomly selected item in candidates
2 east ← furthest item apart from rand // DISTANCE required
3 west ← furthest item apart from east // DISTANCE required
4 c ← DISTANCE(east, west)
5 foreach  $x \in \text{items}$  do
6   a ← DISTANCE(x, west)
7   b ← DISTANCE(x, east)
8    $x.d \leftarrow (a^2 + c^2 - b^2)/(2c)$  // cosine rule
9 sort items by  $x.d$ 
10 eastItems ← first half of items
11 westItems ← second half of items
12 return [west, east], [westItems, eastItems]

```

candidates are then projected onto the line joining the two extreme candidates (line 5-8). Finally, split the candidates into two parts based on their projection in the line.

3.2 SPLIT for binary decision spaces

SWAY using Algorithm 2 performed well on models with numerical decisions (see the POM3 and XOMO results shown below). However, when applied to the problem with binary decisions, i.e. $D = \{0, 1\}^n$, it was observed that SWAY performed far worse than standard MOEAs. On investigation, we found that reducing the decision space into a single line losses some information especially for the binary decisions. Specially, if the candidates are spaced using only the total number of 1-bits, then the distribution of instances was highly skewed towards the lower end.

Algorithm 3: SPLIT for binary decision spaces

```

Input      : items – The candidates to split
Output     : [west, east] – representatives;
               [westItems, eastItems] – two parts
Parameter  : totalGroup – the granularity
Require Func : DISTANCE

1 rand ← randomly selected item in candidates
2 foreach  $x \in \text{items}$  do
3    $x.r \leftarrow |\forall d_i \in x \wedge x_i == 1|$  // sum all the "1" values
4    $x.d \leftarrow \text{DISTANCE}(x, \text{rand})$ 
5 normalize  $x.r$  into  $[0, 1]$ 
6  $R \leftarrow \{i.r \mid i \in \text{items}\}$  // all possible radius
7 foreach  $k \in R$  do
8   // for each possible radius
9   // equally distribute the candidates with k-radius
10  // into the concentric-circle
11   $g \leftarrow \{i \mid i.r = k\}$ 
12  sort  $g$  by  $x.d$ ,  $g \leftarrow x_1, x_2, \dots, x_{|g|}$ 
13  for  $i \in [1, |g|]$  do
14     $x_i.\theta \leftarrow \frac{2\pi i}{|g|}$ 
15 // split the candidates
16  $\text{thk} \leftarrow \max(R) / (\text{totalGroup})$  // the thickness
17 foreach  $a \leq \text{totalGroup}$  do
18   // for the annulus with  $(a-1)\text{thk} \leq \text{radius} \leq a \text{ thk}$ 
19    $g \leftarrow \{i \mid (a-1)\text{thk} \leq i.r \leq a\text{thk}\}$ 
20    $c_1 \leftarrow$  the item with minimum  $\theta$  in  $g$ 
21   add  $c_1$  to east
22    $c_2 \leftarrow$  the item with maximum  $\theta$  in  $g$ 
23   add  $c_2$  to west
24   add items with  $\theta \leq \pi$  in  $g$  to eastItems
25   add items with  $\theta > \pi$  in  $g$  to westItems
26 return [west, east], [westItems, eastItems]

```

Accordingly, inspired by the research in radial basis function kernel [6], we invented a radial co-ordinate system. The radial co-ordinate system can force vectors of binary decisions away from outer edges into the inner volume of that space. Candidates

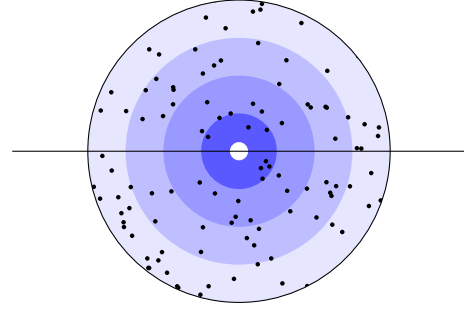


Fig. 2. Map candidates into a circle. The large white dot in the center is the “pivot”, selected randomly among the candidates. All other candidates (the black dots) are located based on their radius and angular coordinates. The circle is divided into multiple equal-thickness annulus. The candidates with minimum angular coordinates form the *east* representatives. The candidates with maximum angular coordinates form the *west* representatives. Candidates whose angular coordinate is less than π (upper semicircle) form the *eastItems* and others (locates in lower semicircle) form the *westItems*.

representing similar-size individuals (share similar # of 1 bits) are grouped and comparisons only be performed inside the group. Algorithm 3 implements such a radial co-ordinate system. This algorithm splits our binary decision using a randomly selected “pivot” point. After that, it maps the other candidates into a *circle*, rather than the *line* showed in Algorithm 2.

To map the candidates into this circle, first, for the candidate $x = (d_0, d_1, \dots, d_n)$ ($d_i \in \{0, 1\}$), we assign $x.r$ as $\sum_{i=0}^n d_i$ and $x.d$ as the *Jaccard distance* between x and the “pivot” candidate (lines 2-4). The Jaccard distance is defined as

$$\sum |a_i - b_i| / (A + B - \sum \min(a_i, b_i)) \quad (A = (a_0, \dots, a_n), B = (b_0, \dots, b_n), a_i, b_i \in \{0, 1\})$$

Once mapped into a circle, we then uniformly spread all candidates with similar r values into a circumference whose radius is r , based on their d values– the one with minimum d values has the minimum angular coordinate; the one with second minimum d values has larger coordinate; and so on (lines 7-11).

This circle is then used to generate the partitions. Figure 2 shows how this circle is divided into several equal-thickness annulus (the number of annulus, i.e., the granularity of SPLIT is a configurable parameter). After the division:

- The candidates with minimum θ in each annulus area form the *east*;
- The candidates with maximum θ form the *west*.
- Candidates in the upper semicircle form the *eastItems* and others form the *westItems*.

3.3 The BETTER function

There are multiply pairs of (east, west) if the problem is with binary decision space while one pair if the problem is with continuous decision space. For each pair, we compare the east representative and west representative. If there are more pairs that east representative dominates west representatives, BETTER returns east half; west half otherwise.

4 CASE STUDIES**4.1 Research Questions**

RQ1: Can SWAY find the solution with maximized (minimized) objective as other MOEAs?

scale factors (exponentially decrease effort)	prec: have we done this before? flex: development flexibility resl: any risk resolution activities? team: team cohesion pmat: process maturity
upper (linearly decrease effort)	acap: analyst capability pcap: programmer capability pcon: programmer continuity aexp: analyst experience pexp: programmer experience ltex: language and tool experience tool: tool use site: multiple site development sced: length of schedule
lower (linearly increase effort)	rely: required reliability data: 2nd memory requirements cplx: program complexity ruse: software reuse docu: documentation requirements time: runtime pressure stor: main memory requirements pvool: platform volatility

Fig. 3. Descriptions of the XOMO variables.

RQ1 is concerned with the quality of SWAY results using the models of §4.2; i.e. XOMO, POM3, and software product lines. For comparison purposes:

- When exploring XOMO and POM3, we also try to optimize this model using NSGA-II and SPEA2. We use NSGA-II and SPEA2 since, according to a survey of the SBSE literature in the period 2004 thru 2013, Sayyad [34] found 25 different algorithms. Of those, NSGA-II [9] or SPEA2 [50] were used four times as often as anything else.
- As to the software product line models, these have been extensively explored in the SBSE literature. Our reading of that work is that the state of the art reported in those papers are the IBEASEED [33] and SATIBEA [18] algorithms (see §4.2.3). Hence, for comparison purposes when optimizing software product lines, we will compare the performance of SWAY to that of IBEASEED [33] and SATIBEA [18].

To assess the performance of these optimisers, we used two measures—hypervolume and spread (defined in §4.3). Also, to check if SWAY results were stable, we repeated the study 20 times and report the median, IQR of their hypervolume and spread (median is the 50th percentile of a list of sorted numbers and the IQR, or inter-quartile range, is the 75th-25th range).

RQ2: To what extent is SWAY faster than the typical MOEAs?

We use the number of model evaluations as the measure of algorithm time-complexity. For all the models of §4.2, we optimize them by different MOEAs and by SWAY. We then report the ratio of the number of evaluations needed by different optimizers.

RQ3: Is SWAY only applicable to the small-size problem?

To answer this question, we compare the performance of SWAY and the other optimizers as the models get progressively more complex. Of the models in §4.2, the XOMO and POM3 models are relatively simple while the software product lines can get very complex indeed.

4.2 Benchmarks

This section, reviews the three SE problems studied in this paper—XOMO, POM3 and software product lines.

project	ranges			values	
	feature	low	high	feature	setting
FLIGHT: JPL's flight software	rely	3	5	tool	2
	data	2	3	sced	3
	cplx	3	6		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
GROUND: JPL's ground software	pmat	2	3		
	KSLOC	7	418		
	rely	1	4	tool	2
	data	2	3	sced	3
	cplx	1	4		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
OSP: Orbital space plane nav& guidance	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		
	prec	1	2	data	3
	flex	2	5	pvool	2
	resl	1	3	rely	5
	team	2	3	pcap	3
	pmat	1	4	plex	3
	stor	3	5	site	3
OSP2: OSP version 2	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
	prec	3	5	flex	3
	pmat	4	5	resl	4
	docu	3	4	team	3
	ltex	2	5	time	3
	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvool	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				pcon	3
				apex	4
				plex	4
				tool	5
				cplx	4
				site	6

Fig. 4. Four project-specific XOMO case studies.

4.2.1 XOMO

XOMO, introduced in [26], is a general framework for Monte Carlo simulations that combines four COCOMO-like software process models from Boehm's group at the University of Southern California. Figure 3 lists the description of XOMO input variables (All should be within [1,6]). The XOMO user begins by defining a set of *ranges* or a specific *value* of these variables to address his or her real situation in one software project. For example, if the project has (a) *relaxed schedule pressure*, they should set *sced* to its minimal value; (b) *reduced functionality*, they should halve the value of *kloc* and minimize the size of the project database (by setting *data*=2); (c) *reduced quality* (for racing something to market), they might move to lowest reliability, minimize the documentation work and the complexity of the code being written, reduce the schedule pressure to some middle value— in the language of XOMO, this last change would be *rely*=1, *docu*=1, *time*=3, *cplx*=1.

XOMO computes four objective scores: (1) project *risk*;

Short name	Decision	Description	Controllable
Cult	Culture	Number (%) of requirements that change.	yes
Crit	Criticality	Requirements cost effect for safety critical systems.	yes
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality.	yes
Init. Kn	Initial Known	Number of (%) initially known requirements.	no
Inter-D	Inter-Dependency	Number of (%) requirements that have inter-dependencies to other teams.	no
Dyna	Dynamism	Rate of how often new requirements are made.	yes
Size	Size	Number of base requirements in the project.	no
Plan	Plan	Prioritization Strategy: 0= Cost Ascending; 1= Cost Descending; 2= Value Ascending; 3= Value Descending; 4= $\frac{Cost}{Value}$ Ascending.	yes
T.Size	Team Size	Number of personnel in each team	yes

Fig. 5. List of inputs to POM3. These inputs come from Turner & Boehm's analysis of factors that control how well organizers can react to agile development practices [5]. The optimization task is to find settings for the controllable in the last column.

(2) development *effort*; (3) predicted *defects*; (4) total *months* of development. Effort and defects are predicted from mathematical models derived from data collected from hundreds of commercial and Defense Department projects [4]. As to the *risk* model, this model contains rules that triggers when management decisions decrease the odds of successfully completing a project: e.g. demanding *more* reliability (*rely*) while *decreasing* analyst capability (*acap*). Such a project is “risky” since it means the manager is demanding more reliability from less skilled analysts. XOMO measures *risk* as the percent of triggered rules.

The optimization goals for XOMO are to:

- Reduce risk;
- Reduce effort;
- Reduce defects;
- Reduce months.

Note that this is a non-trivial problem since the objectives listed above as non-separable and conflicting in nature. For example, *increasing* software reliability *reduces* the number of added defects while *increasing* the software development effort. Also, *more* documentation can improve team communication and *decrease* the number of introduced defects. However, such increased documentation *increases* the development effort.

In our case studies with XOMO, we use four scenarios taken from NASA's Jet Propulsion Laboratory [28]. As shown in Figure 4, FLIGHT and GROUND are general descriptions of all JPL flight and ground software while OSP and OPS2 are two versions of the flight guidance system of the Orbital Space Plane.

4.2.2 POM3– A Model of Agile Development

POM3 model is a tool for exploring the thorny management challenge in agile development [3], [5], [32]– balancing *idle rates*, *completion rates* and *overall cost*. More specifically,

- in the agile world, projects terminate after achieving a *completion rate* of $X\%$ ($X < 100$) of its required tasks;
- team members become *idle* if forced to wait for a yet-to-be-finished task from other teams;

- to lower *idle rate* and improve *completion rate*, management can hire staff—but this *increases overall cost*.

The POM3 model simulates the Boehm and Turner model of agile programming [4] where teams select tasks as they appear in the scrum backlog. Figure 5 lists the inputs of POM3 model. What users feel interested in is how to tune the decisions in order to:

- increase completion rates,
- reduce idle rates,
- reduce overall cost.

One way to understand POM3 is to consider a set intra-dependent requirements. A single requirement consists of a prioritization *value* and a *cost*, along with a list of child-requirements and dependencies. Before any requirement can be satisfied, its children and dependencies must first be satisfied. POM3 builds a requirements heap with prioritization values, containing 30 to 500 requirements, with costs from 1 to 100 (values chosen in consultation with Richard Turner [5]). Since POM3 models agile projects, the *cost*, *value* figures are constantly changing (up until the point when the requirement is completed, after which they become fixed).

Now imagine a mountain of requirements hiding below the surface of a lake; i.e. it is mostly invisible. As the project progresses, the lake dries up and the mountain slowly appears. Programmers standing on the shore study the mountain. Programmers are organized into teams. Every so often, the teams pause to plan their next sprint. At that time, the backlog of tasks comprises the visible requirements.

For their next sprint, teams prioritize work for their next sprint using one of five prioritization methods: (1) cost ascending; (2) cost descending; (3) value ascending; (4) value descending; (5) $\frac{cost}{value}$ ascending. Note that prioritization might be sub-optimal due to the changing nature of the requirements *cost*, *value* as the unknown nature of the remaining requirements. POM3 has another wild card, it contains an *early cancellation probability* that can cancel a project after N sprints (the value directly proportional to number of sprints). Due to this wild-card, POM3's teams are always racing to deliver as much as possible before being re-tasked. The final total cost is a function of:

- (a) Hours worked, taken from the *cost* of the requirements;
- (b) The salary of the developers: less experienced developers get paid less;
- (c) The criticality of the software: mission critical software costs more since they are allocated more resources for software quality tasks.

In our study, we explore three scenarios proposed by Boehm personnel communication (Figure 6). Among them, POM3a covers a wide range of projects; POM3b represents small and highly critical projects and POM3c represent large projects that are highly dynamic, where cost and value can be altered over a large range.

4.2.3 Software Product Lines

A software product line (SPL) is a collection of related software products, which share some core functionality [16]. From one product line, many products can be generated. For example, Apel et al. model the compilation configuration parameters of databases as a product line. By adjusting those configurations, a suite of different database solutions can be generated [37].

Figure 7 shows a feature model for a mobile phone product line. All features are organized as a tree. The relationship between two features might be “mandatory”, “optional”, “alternative”,

	POM3a A broad space of projects.	POM3b Highly critical small projects	POM3c Highly dynamic large projects
Culture	$0.10 \leq x \leq 0.90$	$0.10 \leq x \leq 0.90$	$0.50 \leq x \leq 0.90$
Criticality	$0.82 \leq x \leq 1.26$	$0.82 \leq x \leq 1.26$	$0.82 \leq x \leq 1.26$
Criticality Modifier	$0.02 \leq x \leq 0.10$	$0.80 \leq x \leq 0.95$	$0.02 \leq x \leq 0.08$
Initial Known	$0.40 \leq x \leq 0.70$	$0.40 \leq x \leq 0.70$	$0.20 \leq x \leq 0.50$
Inter-Dependency	$0.0 \leq x \leq 1.0$	$0.0 \leq x \leq 1.0$	$0.0 \leq x \leq 50.0$
Dynamism	$1.0 \leq x \leq 50.0$	$1.0 \leq x \leq 50.0$	$40.0 \leq x \leq 50.0$
Size	$x \in [3, 10, 30, 100, 300]$	$x \in [3, 10, 30]$	$x \in [30, 100, 300]$
Team Size	$1.0 \leq x \leq 44.0$	$1.0 \leq x \leq 44.0$	$20.0 \leq x \leq 44.0$
Plan	$0 \leq x \leq 4$	$0 \leq x \leq 4$	$0 \leq x \leq 4$

Fig. 6. Three specific POM3 scenarios.

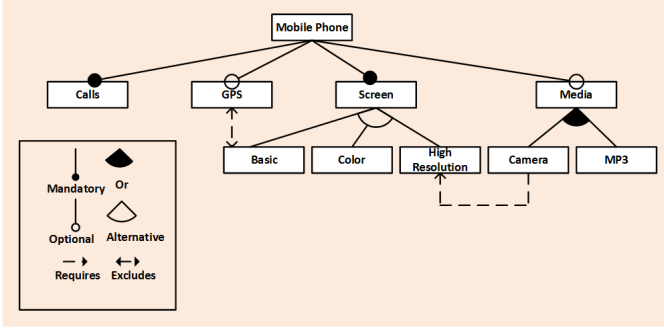


Fig. 7. Feature model for mobile phone product line. To form a mobile phone, “Calls” and “Screen” are the mandatory features (shown as solid •), while the “GPS” and “Media” features are optional (shown as hollow ○). The “Screen” feature can be “Basic”, “Color” or “High resolution” (the alternative relationship). The “Media” feature contains “camera”, “MP3”, or both (the Or relationship).

or “or”. Also, there exists some cross-tree constraints, which means the referred features are not in the same sub-tree. These cross-tree constraints complicate the process of exploring feature models². In practice, all constraints, including the tree-structure constraints and the cross-tree constraints can be expressed by the CNF (conjunctive normal form). For example, Figure 7 indicates the following 19 CNFs.

- ¬Mobile Phone ∨ Calls
- Mobile Phone ∨ ¬Calls
- ¬Mobile Phone ∨ Screen
- Mobile Phone ∨ ¬Screen
- Mobile Phone ∨ ¬GPS
- Mobile Phone ∨ ¬Media
- Media ∨ ¬Camera
- Media ∨ ¬MP3
- ¬Media ∨ Camera ∨ MP3
- Screen ∨ ¬Basic
- Screen ∨ ¬Color
- Screen ∨ ¬High resolution
- ¬Screen ∨ Basic ∨ Color ∨ High resolution
- ¬Basic ∨ ¬Color ∨ ¬High resolution
- Basic ∨ ¬Color ∨ ¬High resolution
- ¬Basic ∨ Color ∨ ¬High resolution
- ¬Basic ∨ ¬Color ∨ High resolution
- ¬GPS ∨ ¬Basic
- ¬Camera ∨ High resolution

In our case studies, the optimization of product generation from software product lines is a five goal optimization problem:

- 1) Find the valid products (products not violating any cross-tree constraint or tree structure) which have..
- 2) More features; and
- 3) Less known defeats; and
- 4) Less total cost; and
- 5) Most features used in prior applications.

2. Without cross-tree constraints, one can explore the feature model through the top-down breath-first search.

Following the recommendation of Sayyad [33], defect, cost, and knowledge of usage in prior applications is set stochastically.

A major problem with analyzing software product lines is that, when the cross-tree constraints get complex, it can be very hard to find valid products. This above set of constraints is relatively simple but real-world software product lines can be far more complex. As shown in Figure 8, software product line models comprise up to tens of thousands of features, with 100,000s of constraints. These networks of constraints can get so complex that random assignments of “use” or “do not use” to the features have very low probability of satisfying the constraints. For example, in one of our software product lines, the *Linux* model, we generated 10,000 random sets of decisions for the features. Within that space, less than 5 decisions were valid.

Consequently, much of the research on optimizing the generation of products from software product lines has focused on how best to optimize within this heavily-constrained models:

- Sayyad et al. [33] introduced the *IBEASEED* method– a five-goal optimization problem had its first generation of candidates initialized by a pre-processor that just sought out valid products (and one other goal).
- *SATIBEA* was introduced by Henard et al. [18]. This makes full use of SAT solver technology to fix the invalid candidates every time the “mutate” or “crossover” operation is performed in the IBEA algorithm. Results showed that the SATIBEA algorithm can find the valid products for the extremely large feature models by tens of thousands evaluations, much better than other algorithms.
- In the case studies described below, when SWAY explores software product lines, it first satisfies goal #1 (find valid products) by running the SAT4j SAT solver over the product line to generate valid products. SAT solvers like SAT4j are highly optimized programs that can find valid solutions with CNFs. For more details, on SAT4j see <http://www.sat4j.org/>.

To the best of our knowledge, the *IBEASEED* and *SATIBEA* methods are the best two algorithms to solve software product line optimization problem. Consequently, for product line models we compare the performance of SWAY to these two algorithms.

4.3 Performance Measures

Number of evaluations: To compare the runtime of different algorithms, we use the number of model evaluations as our metric. This study did not use absolute runtime since in actual practice, the software engineering models are extremely large, and the evaluation process occupies significant part of the runtime. Also, various implementation languages (e.g. Java, Python, C++, etc.) or compilers influence the absolute runtime.

To assess the quality of results, we use two metrics, spread and hypervolume, both of which are adopted by many papers

Database	Name	Number of Features	Constraints
SPLOT	webportal	49	81
	eshop	330	506
LVAT	toybox	544	1020
	uClinux	1850	2468
	ecos	1244	3146
	fiasco	1638	5228
	coreboot	12268	47091
	freebsd	1396	62183
	embtoolkit	23516	180511
	linux	6888	343944

Fig. 8. Feature models used in this study, sorted by the number of constraints. The constraints includes the tree-structure and cross-tree constraints. SPLOT models can be found at <http://www.splot-research.org/> and LVAT models are at <https://code.google.com/archive/p/linux-variability-analysis-tools/>

[16], [33], [35], [49]. Spread and hypervolume are two quality indicators for the space under the final Pareto frontier created by the different optimizers. As in [49], we apply binary domination to the output of our optimizers to define the final frontier.

As defined in [9], **Spread** measures the extent of spread in the frontier. Formally,

$$\text{Spread} = \frac{d_f + d_l + \sum_{i=1}^N |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}} \quad (3)$$

where N is the number of individuals in the frontier; d_i is the Euclidean distances between consecutive points; \bar{d} is the average of d_i s, and the d_f and d_l are the Euclidean distance between the extreme solutions and the boundary solutions of obtained frontier.

A “good” spread makes all the distance equal, i.e., the individuals are equally distributed in the frontier.

As defined in [51], **Hypervolume** measures the size of space the obtained frontier covered. Formally,

$$\text{Hypervolume} = \lambda \left(\bigcup_{y \in B} \{y' | y < y' < y_{\text{ref}}\} \right) \quad (4)$$

where

- $\lambda(\cdot)$ is the Lebesgue measure, the standard way measures the subset of n -dimensional Euclidean space. For example, Lebesgue measure is the length, area or volume when $n = 1, 2, 3$ respectively;
- B is the obtained final frontier;
- $<$ is the binary domination comparator;
- y_{ref} denotes a reference point that should be dominated by all obtained solutions.

Note that, in this study, all objectives are normalized to $[0, 1]$ and set $y_{\text{ref}} = (1, 1, \dots, 1)$.

According the definitions, *higher* values of hypervolume are *better* while *lower* values of spread are *better*.

To test the performance robustness and reduce the observational error, we repeated these case studies 20 times. For each scenario, each algorithm, we record the median as well as the IQR value of the metrics (number of evaluations, the spread and hypervolume of the obtained frontier) from the 20 repeats.

All results are studied using nonparametric statistics. The use of such nonparametrics for SBSE was recently endorsed by Arcuri and Briand at ICSE’11 [29]. For testing statistical significance, we used nonparametric bootstrap test 95% confidence [10] followed by an A12 test to check that any observed differences were not trivially small effects. Given two lists X and Y , A12 counts how

often there are larger numbers in the former list (for there there are ties, add a half mark). That is,

$$a = \frac{|\{(x, y) | x > y\}| + 0.5 \cdot |\{(x, y) | x = y\}|}{|X| \cdot |Y|}, (x, y) \in X \times Y$$

As per Vargha [42], we say that a “small” effect has $a < 0.6$.

Lastly, to rank our optimizers, we use the Scott-Knott test to recursively divide our optimizers. This recursion uses A12 and bootstrapping to group together subsets that are (a) not significantly different and are (b) not just a small effect different to each other. This use of Scott-Knott is endorsed by Mittas and Angelis in their recent 2013 TSE article [29] and by Hassan et al. in their recent 2015 ICSE article [14].

4.4 Case Study Configurations

Table 1 and Table 2 show the configuration parameters used in these case studies. SWAY-specific parameters are set in the row “SWAY config”. For an explanation of these parameters, see the above algorithm listings. As to the other optimizers, we followed all parameters in the original papers, with one exception: unlike the original papers which were set the maximum running time, in this study, we set the maximum evaluations as our termination condition. Specifically, because of the extra evaluations in last generation, the number of evaluations might be larger than the maximum threshold, but no larger by the number of population size.

Our case studies explore models with different kinds of decisions. Accordingly:

- We use Algorithm 2 for SWAY’s processing of models with continuous decisions (POM3, XOMO),
- We use Algorithm 3 for SWAY’s processing of models with binary decisions (software product lines).

Our case studies use different methods to generate the initial population:

- IBEASEED and SATIBEA have their own methods for such generation (described above).

TABLE 1
Parameter configurations for XOMO and POM3 scenarios

XOMO/POM3	NSGA-II	SPEA2	SWAY
Initial population source	randomly generated within scenario domain		
Population size	100	100	10,000
Mutation rate	0.01	0.01	na
Crossover rate	0.9	0.9	na
Maximum evaluations	2000		na
SWAY config	na	na	enough= $\sqrt{10,000} = 100$
Repeated runs	20		

na = not applicable

TABLE 2
Parameter configurations for software product line scenarios

Software product line	IBEASEED	SATIBEA	SWAY
Initial population source	randomly generated		SAT4j*
Initial population	300	300	10000
Archive size	300	300	na
Mutation rate	0.001	standard mutate: 0.001 smart mutate: 0.98	na
Crossover rate	0.05	0.05	na
SWAY config	na	na	enough=100 group#= $\frac{1}{3} D $
Maximum Evaluations	50000		na
Repeated runs	20		

*apply random literal selection strategy
|D| is the dimension of decision spaces
na = not applicable

a. Spread (less is better)					b. Hypervolume (more is better)					c. # evaluations (less is better)				
Rank	using	med.	IQR		Rank	using	med.	IQR		Rank	using	med.	IQR	
Flight					Flight					Flight				
1	SWAY	100	18	●	1	NSGA-II	105	1	●	1	SWAY	68	2	●
1	NSGA-II	109	9	●	1	SPEA 2	105	1	●	2	SPEA 2	2007	13	●
1	SPEA 2	109	9	●	2	SWAY	100	4	●	2	NSGA-II	2015	42	●
Ground					Ground					Ground				
1	SWAY	100	9	●	1	NSGA-II	105	1	●	1	SWAY	66	5	●
1	NSGA-II	100	14	●	1	SPEA 2	104	1	●	2	SPEA 2	2037	9	●
1	SPEA 2	106	12	●	2	SWAY	100	6	●	2	NSGA-II	2075	10	●
OSP					OSP					OSP				
1	SWAY	100	12	●	1	NSGA-II	106	0	●	1	SWAY	58	5	●
1	SPEA 2	100	6	●	1	SPEA 2	106	0	●	2	NSGA-II	2005	19	●
1	NSGA-II	101	6	●	2	SWAY	100	3	●	2	SPEA 2	2027	23	●
OSP2					OSP2					OSP2				
1	SWAY	100	6	●	1	NSGA-II	104	1	●	1	SWAY	55	2	●
2	SPEA 2	110	12	●	1	SPEA 2	104	0	●	2	SPEA 2	2007	13	●
3	NSGA-II	126	6	●	2	SWAY	100	6	●	2	NSGA-II	2088	36	●
Pom3a					Pom3a					Pom3a				
1	SWAY	100	12	●	1	SPEA 2	102	0	●	1	SWAY	56	6	●
2	NSGA-II	145	12	●	1	NSGA-II	102	0	●	2	SPEA 2	2001	6	●
2	SPEA 2	159	16	●	2	SWAY	100	0	●	2	NSGA-II	2009	4	●
Pom3b					Pom3b					Pom3b				
1	SWAY	100	15	●	1	NSGA-II	147	19	●	1	SWAY	55	2	●
2	NSGA-II	132	16	●	1	SPEA 2	134	6	●	2	SPEA 2	2003	7	●
2	SPEA 2	140	10	●	2	SWAY	100	7	●	2	NSGA-II	2006	3	●
Pom3c					Pom3c					Pom3c				
1	SWAY	100	9	●	1	NSGA-II	102	0	●	1	SWAY	62	2	●
2	NSGA-II	149	32	●	1	SPEA 2	102	0	●	2	SPEA 2	2010	6	●
2	SPEA 2	170	26	●	2	SWAY	100	1	●	2	NSGA-II	2014	8	●

Fig. 9. Results from POM3 and XOMO. Spread (left), hypervolumes (middle) and number of evaluations (right) seen in 20 independent runs. For convenience, the spread and hypervolume results are scaled to the median of “SWAY” values in each scenario (let median of SWAY scenario be 100). **med.** is the 50th percentile and **IQR** is the interquartile range, i.e. 25th-75th percentile. Lines with a dot in the middle (e.g. —●—) show the median as a round dot within the IQR (and if the IQR is vanishingly small, only a round dot will be visible). Also, all results sorted by the median value— spread and number of model evaluation results are sorted ascending (since *less* is *better*) while hypervolume results are sorted descending (since *more* is *better*). The left-hand side columns **Rank**(see text for the ranking criteria) the optimizers. Red dots “●” denote median results that are “reasonable close” to the top-ranked result (see text for details).

a. Spread (less is better)					b. Hypervolume (more is better)					c. # evaluations (less is better)				
Rank	using	med.	IQR		Rank	using	med.	IQR		Rank	using	med.	IQR	
webportal					webportal					webportal				
1	SATIBEA	0.63	0.05	●	1	SATIBEA	0.33	0.01	●	1	SWAY	150	15	●
2	IBEAseed	0.64	0.04	●	2	IBEAseed	0.31	0.01	●	2	IBEAseed	50071	21	●
2	SWAY	0.66	0.12	●	2	SWAY	0.28	0.03	●	2	SATIBEA	50102	18	●
3	baseline	0.85	0.2	●	3	baseline	0.19	0.05	●	—	baseline	no stat		
eshop					eshop					eshop				
1	SATIBEA	0.63	0.05	●	1	SATIBEA	0.3	0.0	●	1	SWAY	405	6	●
1	SWAY	0.65	0.03	●	2	SWAY	0.26	0.0	●	2	IBEAseed	50178	43	●
2	baseline	0.81	0.25	●	3	baseline	0.16	0.05	●	2	SATIBEA	50201	33	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
toybox					toybox					toybox				
1	SWAY	0.69	0.03	●	1	SWAY	0.19	0.0	●	1	SWAY	231	8	●
1	SATIBEA	0.76	0.08	●	1	SATIBEA	0.19	0.0	●	2	IBEAseed	50033	39	●
2	baseline	0.85	0.05	●	2	baseline	0.18	0.0	●	2	SATIBEA	50102	41	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
uClinux					uClinux					uClinux				
1	SATIBEA	0.68	0.06	●	1	SATIBEA	0.24	0.0	●	1	SWAY	203	4	●
2	baseline	0.92	0.15	●	2	SWAY	0.21	0.0	●	2	SATIBEA	50016	45	●
2	SWAY	0.94	0.04	●	3	baseline	0.2	0.04	●	2	IBEAseed	50105	56	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
ecos					ecos					ecos				
1	SWAY	0.61	0.0	●	1	SATIBEA	0.35	0.0	●	1	SWAY	1207	8	●
1	SATIBEA	0.61	0.08	●	1	SWAY	0.35	0.0	●	2	IBEAseed	50111	36	●
2	baseline	0.85	0.16	●	2	baseline	0.19	0.07	●	2	SATIBEA	50132	45	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
fiasco					fiasco					fiasco				
1	SWAY	0.67	0.01	●	1	SWAY	0.17	0.0	●	1	SWAY	780	13	●
2	SATIBEA	0.84	0.05	●	1	SATIBEA	0.16	0.0	●	2	IBEAseed	50071	23	●
3	baseline	0.96	0.04	●	2	baseline	0.15	0.02	●	2	SATIBEA	50112	22	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
coreboot					coreboot					coreboot				
1	SATIBEA	0.69	0.09	●	1	SWAY	0.27	0.0	●	1	SWAY	2034	12	●
1	SWAY	0.71	0.01	●	1	SATIBEA	0.26	0.0	●	2	IBEAseed	50071	26	●
2	baseline	0.84	0.19	●	2	baseline	0.25	0.01	●	2	SATIBEA	50109	35	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
freebsd					freebsd					freebsd				
1	SWAY	0.75	0.1	●	1	SWAY	0.27	0.0	●	1	SWAY	780	4	●
1	baseline	0.84	0.14	●	2	baseline	0.23	0.04	●	2	SATIBEA	50102	22	●
2	SATIBEA	0.87	0.28	●	3	SATIBEA	0.2	0.12	●	2	IBEAseed	50201	36	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
embtoolkit					embtoolkit					embtoolkit				
1	SWAY	0.60	0.0	●	1	SWAY	0.23	0.0	●	1	SWAY	3704	14	●
1	SATIBEA	0.70	0.05	●	1	SATIBEA	0.22	0.01	●	2	IBEAseed	50121	16	●
2	baseline	0.86	0.07	●	2	baseline	0.21	0.01	●	2	SATIBEA	50122	35	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		
linux					linux					linux				
1	SATIBEA	0.70	0.13	●	1	SWAY	0.30	0.0	●	1	SWAY	1900	11	●
1	SWAY	0.71	0.03	●	2	SATIBEA	0.28	0.0	●	2	SATIBEA	50113	7	●
1	baseline	0.74	0.32	●	3	baseline	0.20	0.03	●	2	IBEAseed	50203	63	●
n/a	IBEAseed	n/a			n/a	IBEAseed	n/a			—	baseline	no stat		

Fig. 10. Software product line results. Same format as Figure 9. “n/a” indicates that there were less than 5 valid products.

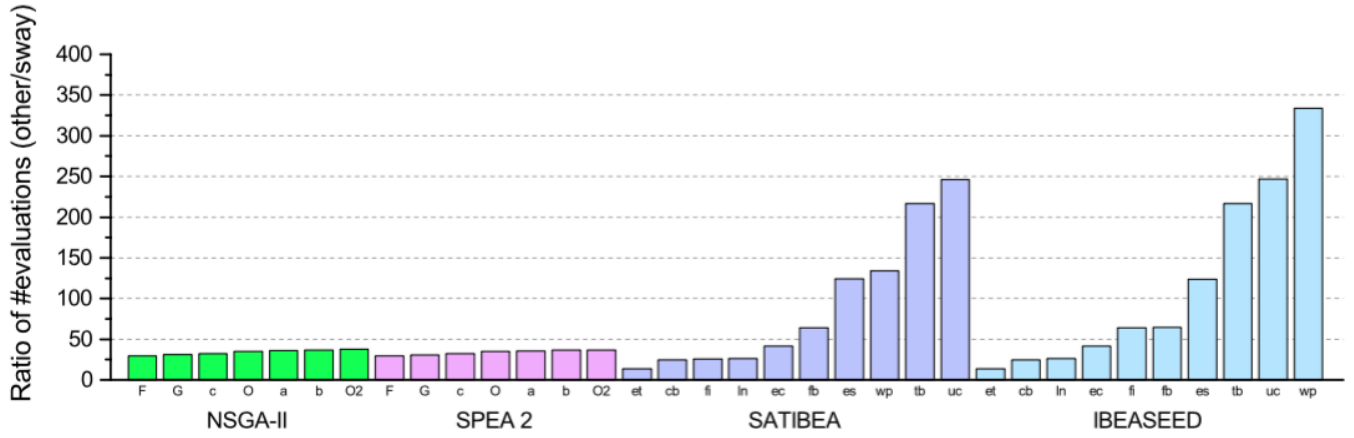


Fig. 11. Ratio of the number of evaluations needed by other optimizers, compared to SWAY. The NSGA-II and SPEA2 results from Figure 9 and the SAIBEA and IBEASEED results come from Figure 10; F, G, O, O2, a, b, c = Flight, Ground, OSP, OP2, POM3a, POM3b, POM3c; wp, es, tb, uc, ec, fi, cb, fb, et, ln = webportal, eshop, toybpx, uClinux, ecos, fiasco, coreboot, freebsd, embtoolkit, linux. Note that these ratios are very large (30 to 300); i.e. SWAY can optimize even complex models one to two orders of magnitude faster than other algorithms.

- For models with few cross-tree constraints (POM3, XOMO), SWAY uses random generation.
- For heavily constrained models (software product lines), SWAY uses a SAT solver to generate the population.

Since our algorithms make use of stochastic decisions, we repeated our case studies 20 times.

Shepperd and MscDonell [36] argue that measurements taken from some complex method should be baselined against measurements taken from some simpler “dumber” method. Accordingly, in our results, we report all valid individuals found by a very limited run of SATIBEA (just 10 generations; i.e. $\frac{1,000}{50,000} = 2\%$ of its execution).

5 RESULTS

Figure 9 shows the results of XOMO and POM3 case studies. For a definition of the scenarios *OSP*, *GROUND*, *OSP2*, *FLIGHT*, *Pom3a*, *Pom3b*, *Pom3c*, see Figure 4 and Figure 5.³

Figure 10 shows the results from the software product line cast study. Note that these models are sorted top-to-bottom least-to-most complicated. For example, as shown in Figure 8, the top-most model (*webportal*) has orders of magnitude fewer features and constraints than the lower models (e.g. *embtoolkit*).⁴

All these results present the median, and interquartile range (the 75th-25th percentiles) seen across 20 repeated runs (with different random number seeds). Also, the results are sorted by their median value and then ranked in column 1. An optimizer is ranked 1 (**Rank**=1) if other optimizers have (a) worse medians; and (b) the other distributions are significantly different (computed via Scott-Knott and bootstrapping); and (c) differences are not a small effect (computed via A12).

The red dots in Figure 9 show where we balance statistical results with some engineering judgment. For example, consider the hypervolume of *Flight* scenario: here SWAY is ranked second even though its value is just $(\frac{105-100}{100} = 5\%)$ worse than that of NSGA-II. Here, we are reluctant to deprecate SWAY’s results since it achieves results very close to NSGA-II and does so using

$\frac{2015}{68} \approx 30$ times fewer evaluations. Hence, we mark that result “reasonably close” to the top-ranked result.

Turning now to our research questions:

RQ1: Can SWAY find optimizations as good as other MOEAs?

In the XOMO and POM3 results in Figure 9, SWAY’s spreads are as good, and sometimes even better than traditional evolutionary optimizers. As to hypervolume, in most cases, SWAY’s hypervolumes are reasonably close to those other optimizers (the exception being *POM3b*).

In the software product line models results of Figure 10, in most cases, SWAY’s spreads are top-ranked or reasonably close to the top ranked optimizer (the exception being *uClinux*). As to the hypervolume results, SWAY once again fails for *uClinux*. That said, in the majority case ($\frac{7}{11} \approx \frac{2}{3}$) SWAY is top-ranked. Also, in two cases of the remaining cases, SWAY’s results are reasonable close to first rank.

From the above, we conclude that measured in terms of hypervolume and spread, sampling a large population of solutions with SWAY is competitive with traditional evolutionary algorithms.

RQ2: To what extent is SWAY faster than the typical MOEAs? The third column in Figure 9 and Figure 10 list the number of evaluations required by SWAY and the other optimizers. Figure 11 expresses those numbers as ratios of the number of SWAY evaluations. Note that all those ratios are more than one; i.e. SWAY always does what it does with fewer evaluations. Further, those ratios range for 14 to 336; i.e. sampling with SWAY achieves its results using orders of magnitude fewer evaluations than traditional evolutionary algorithms.

RQ3: Is SWAY only applicable to the small-size problem?

The above results suggest that the *more* complex the model, the *greater* the advantage of sampling with SWAY over traditional evolutionary algorithms:

- In Figure 10, there were $\frac{7}{11} \approx 64\%$ models where SWAY had a top-ranked hypervolume. Note that for the top six most complex software line models (shown at the bottom of Figure 10), that ratio is $\frac{6}{6} = 100\%$.
- In Figure 11, the largest ratio gains for SWAY come from the software product line models (the two groups of results on the right hand side of that figure).

3. Source code for XOMO and POM3 can be downloaded at <http://bit.ly/2bJMywS>.

4. Source code for software product line cast study can be downloaded at <http://bit.ly/2bE9hgr>.

That is, for sampling, the *harder* the model, the *better* the performance compared to evolutionary methods.

6 THREATS TO VALIDITY

6.1 Optimizer bias

There are theoretical reasons to conclude that it is impossible to show that any one optimizer *always* performs best. Wolpert and Macready [46] showed in 1997 that no optimizers necessarily work better than any other for all possible optimization problems⁵.

In this study, we compared SWAY framework with NSGA-II, SPEA2 (for XOMO and POM3 cast studies) and modified IBEA (for software product line case studies). We selected those learners since:

- The literature review of Sayyad et al. [33] reported that NSGA-II and SPEA2 were widely used in the SBSE literature;
- In the particular case of software product lines, we have seen that the IBEA variants are widely recognized as being the state of the art.

That said, there exist many other optimizers. For examples, NSGA-III is an improved version of NSGA-II, which can get better diversity of the results. Such kind of optimizers might perform better than SWAY method. Also, for some specific problem, researchers might propose some modified version of MOEAs. For example, [19] is an improved algorithm to solve the software product line problem whose modules are organized as the feature tree, instead of the raw CNF in this study. For such kind of modified/optimized algorithms, SWAY might not perform as well as them.

6.2 Sampling bias

We used the XOMO, POM3 and software product line optimization problems for our case studies. We found that in all of these problems, SWAY can perform as well (or better) as other MOEAs. However, there are many other optimization problems in the area of software engineering.

It is very difficult to find the representatives sample of models which covers all kinds of models. Some problems might have other properties which make it different from any problem we tested in this study. For example, project planning problem is an open question up to now. In the project planning problem, there are constraints which organized as the directed acyclic graph(DAG). None of problems we tested with such kind of constraints.

For this issue of sampling bias (and for optimizer bias), we cannot explore *all* problems here. What we can do:

- Detail our current work and encourage other researchers to test our software on a wider range of problems;
- In future work, apply SWAY when we come across a new problem and compare them to the existed algorithms.

6.3 Evaluation bias

We evaluated the results through spread and hypervolume and number of evaluations. There are many other measures which are adopted in the community of software engineering. For example, a widely used evaluation measures for the multi-objective optimization problem is the inverted generational distance (*igd*) indicator [41].

⁵. “The computational cost of finding a solution, averaged over all problems in the class, is the same for any solution method. No solution therefore offers a short cut.” [46]

Using various measures might lead to different conclusions. This threatens our conclusion. A comprehensive analysis using other measures is left to the future work.

7 CONCLUSION

SWAY is a sampling technique—it can find the most promising individuals among a large set of candidates only with very limited model evaluations. Since the number of required model evaluations is very small, SWAY can run very fast—particularly for large, heavily-constrained models. It would be a very useful approach for solving the complex SE problem whose evaluations are time-consuming and/or expensive.

Unlike traditional evolution-based algorithms, SWAY does not have mutation and crossover operations. Instead it generate large initial population (either randomly, or using a SAT solver), then explore that sample looking for the best subsets.

Prior to this paper, we had thought that this sampling could be usefully augmented with multi-generational mutation and crossover. That line of thinking lead to the GALE algorithm [21]–[24]. Based on the results of this paper, we must now report that multi-generational mutation and crossover seems to add little value over sampling a large initial population. Further, as discussed in our answer to **RQ3**, it seems that sampling works relatively *better* (compared to evolution), the *harder* the optimization problem. Hence, our research plan is to revisit many of the prior results SBSE results based on evolution to see in sampling can solve those faster and/or better.

Why does SWAY work so well? To answer that, we turn to the machine learning literature where Dasgupta and Freund [8] comment:

A recent positive in that field has been the realization that a lot of data which superficially lie in a very high-dimensional space R^D , actually have low intrinsic dimension, in the sense of lying close to a manifold of dimension $d \ll D$.

One way to discover those lower dimensions are random projection methods [8], which include the polar co-ordinate system used in this paper. Hence, our conjecture is that SWAY works so well due to the inherent low intrinsic dimensionality of the models explored in the paper. An open issue here is “how many seemingly complex SE problems are, in fact, low-dimensional?”. If low-dimensionality is common, the tools like SWAY could be widely applicable. This possibility needs to be checked via further research.

To conclude this paper, we speculate on the most glaring question raised by this work:

If simple sampling methods (like SWAY) work so well, why was this not discovered earlier?

We have no definitive answer, except for the following comment. It seems to us that the culture of modern SE research rewards complex elaboration rather than the careful reconsideration and simplification of existing techniques,. Perhaps it is time to reconsider the old saying “if it ain’t broke, don’t fix it”. Our revision to this saying might be “if it ain’t broke, keep cutting something till it breaks”. The results of this paper suggest that such “cutting” can lead to startling and useful results that challenge decades-old beliefs.

REFERENCES

- [1] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction*. Morgan Kaufmann Publishers San Francisco, 1998.
- [2] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.
- [3] B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.
- [4] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
- [5] Barry Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co., Inc., 2003.
- [6] Kai-Min Chung, Wei-Chun Kao, Chia-Liang Sun, Li-Lun Wang, and Chih-Jen Lin. Radius margin bounds for support vector machines with the rbf kernel. *Neural computation*, 15(11):2643–2681, 2003.
- [7] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.
- [8] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *40th ACM Symposium on Theory of Computing*, 2008.
- [9] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International Conference on Parallel Problem Solving From Nature*, pages 849–858. Springer, 2000.
- [10] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC, 1993.
- [11] Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD international conference on Management of data*, 1995.
- [12] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. Not going to take this anymore: multi-objective overtime planning for software engineering projects. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 462–471. IEEE Press, 2013.
- [13] Wei Fu, Tim Menzies, and Xipeng Shen. Tuning for Software Analytics: is it Really Necessary? *Information and Software Technology*, 2016.
- [14] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *37th IEEE International Conference on Software Engineering*, May 2015.
- [15] M. Harman. Personal communication, 2013.
- [16] Mark Harman, Yue Jia, Jens Krinke, William B Langdon, Justyna Petke, and Yuanyuan Zhang. Search based software engineering for software product line engineering: a survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 5–18. ACM, 2014.
- [17] Mark Harman, Phil McMinn, Jerffeson Teixeira De Souza, and Shin Yoo. Search Based Software Engineering: Techniques, Taxonomy, Tutorial. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7007:1–59, 2012.
- [18] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 517–528. IEEE, 2015.
- [19] Robert M Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):17, 2016.
- [20] John H. Holland. Genetic Algorithms. *Scientific American*, 267(1):66–72, 1992.
- [21] J. Krall, T. Menzies, and M. Davies. Gale: Geometric active learning for search-based software engineering. *IEEE Transactions on Software Engineering*, 41(10):1001–1018, Oct 2015.
- [22] J. Krall, T. Menzies, and M. Davies. Learning mitigations for pilot issues when landing aircraft (via multiobjective optimization and multiagent simulations). *IEEE Transactions on Human-Machine Systems*, 46(2):221–230, April 2016.
- [23] Joseph Krall. *Faster Evolutionary Multi-Objective Optimization via GALE, the Geometric Active Learner*. PhD thesis, West Virginia University, 2014. <http://go.ggl/u8ganF>.
- [24] Joseph Krall, Tim Menzies, and Misty Davies. Learning the task management space of an aircraft approach model. In *Proceedings of the 2014 AAAI Conference, AAAI’14*, 2014.
- [25] Tim Menzies, Oussama El-Rawas, J. Hihn, M. Feather, B. Boehm, and R. Madachy. The business case for automated software engineering. In *ASE ’07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 303–312. New York, NY, USA, 2007. ACM.
- [26] Tim Menzies and Julian Richardson. Xomo: Understanding development options for autonomy. In *COCOMO forum*, volume 2005, 2005.
- [27] Tim Menzies, S. Williams, Oussama El-Rawas, D. Baker, B. Boehm, J. Hihn, K. Lum, and R. Madachy. Accurate estimates without local data? *Software Process Improvement and Practice*, 14:213–225, July 2009.
- [28] Tim Menzies, S. Williams, Oussama El-Rawas, B. Boehm, and J. Hihn. How to avoid drastic software process change (using stochastic stability). In *ICSE’09*, 2009.
- [29] Nikolaos Mittas and Lefteris Angelis. Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm. *IEEE Transactions of Software Engineering*, 2013.
- [30] A.J. Nebro, J.J. Durillo, J. Garcia-Nieto, C.A. Coello Coello, F. Luna, and E. Alba. SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In *Computational intelligence in multi-criteria decision-making, 2009. mcdm ’09. iee symposium on*, pages 66–73, March 2009.
- [31] John C. Platt. Fastmap, metricmap, and landmark MDS are all nystrom algorithms. In *In Proceedings of 10th International Workshop on Artificial Intelligence and Statistics*, pages 261–268, 2005.
- [32] D. Port, A. Olkov, and T. Menzies. Using simulation to investigate requirements prioritization strategies. In *Automated Software Engineering*, pages 268–277, 2008.
- [33] A. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel’s back. In *ASE’13, Palo Alto, CA*, 2013.
- [34] Abdel Sayyad and Hany Ammar. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2013.
- [35] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. In *Software engineering (ICSE), 2013 35th international conference on*, pages 492–501. IEEE, 2013.
- [36] Martin Shepperd and Steve MacDonell. Evaluating prediction systems in software project estimation. *Information and Software Technology*, 54(8):820 – 827, 2012. Special Issue: Voice of the Editorial Board/Special Issue: Voice of the Editorial Board.
- [37] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, pages 167–177. IEEE Press, 2012.
- [38] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
- [39] Chakrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *38th International Conference on Software Engineering*, 2016.
- [40] T. Menzies V. Nair and J.Chen. An (accidental) exploration of alternatives to evolutionary algorithms for sbse. In *SSBSE’16*, 2016.
- [41] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Citeseer, 1998.
- [42] András Vargha and Harold D Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 2000.
- [43] Varsha Veerappa and Emmanuel Letier. Understanding clusters of optimal solutions in multi-objective decision problems. In *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*, pages 89–98, 2011.
- [44] Shuai Wang, Shaikat Ali, and Arnaud Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1493–1500. ACM, 2013.
- [45] Tiantian Wang, Mark Harman, Yue Jia, and Jens Krinke. Searching for better configurations: A rigorous approach to clone evaluation. In *9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013.
- [46] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, Apr 1997.
- [47] Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE transactions on evolutionary computation*, 2007.
- [48] Yuanyuan Zhang, Mark Harman, and Afshin Mansouri. The sbse repository: A repository and analysis of authors and research articles on search based software engineering.
- [49] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *International Conference on Parallel Problem Solving from Nature*, pages 832–842. Springer, 2004.
- [50] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, 2002.
- [51] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on evolutionary computation*, 3(4):257–271, 1999.
- [52] Marcela Zuluaga, Andreas Krause, Guillaume Sergent, and Markus Püschel. Active learning for multi-objective optimization. In *International Conference on Machine Learning (ICML)*, 2013.